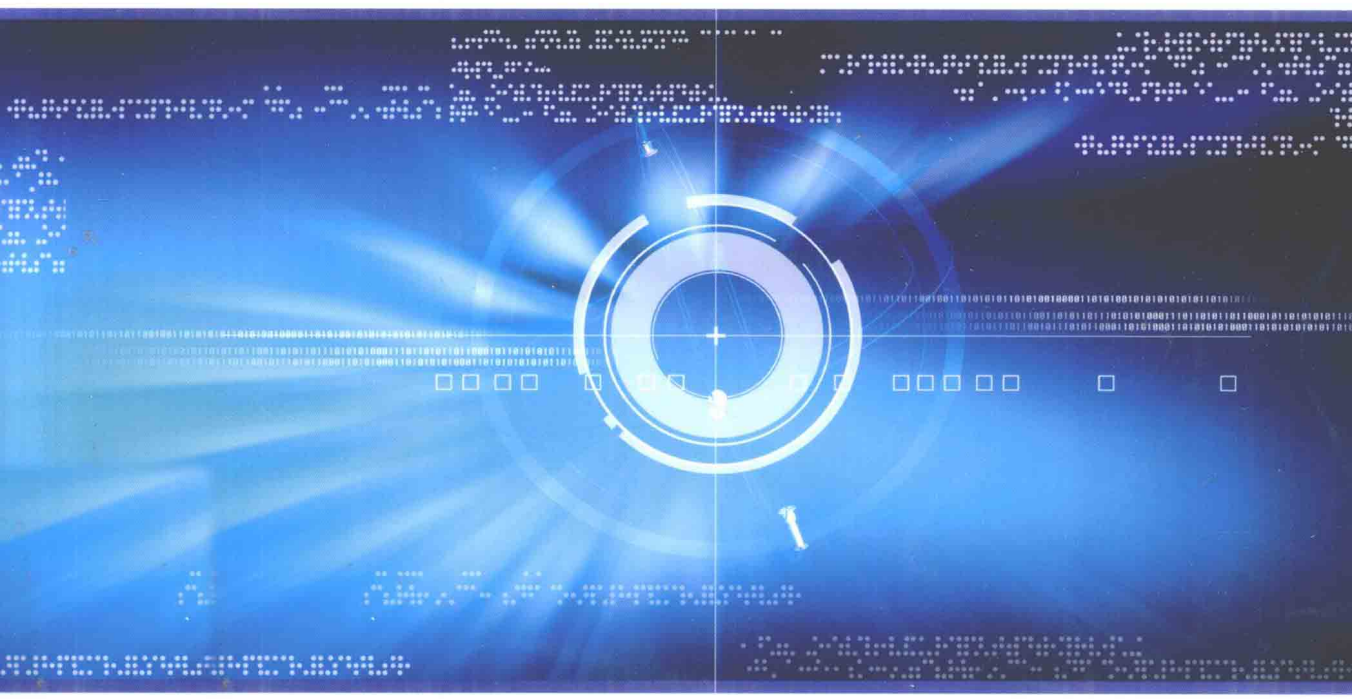
HZ BOOKS
华章教育

高等院校计算机教材系列

网络编程与分层协议设计

基于Linux平台实现

刘 飚 编著



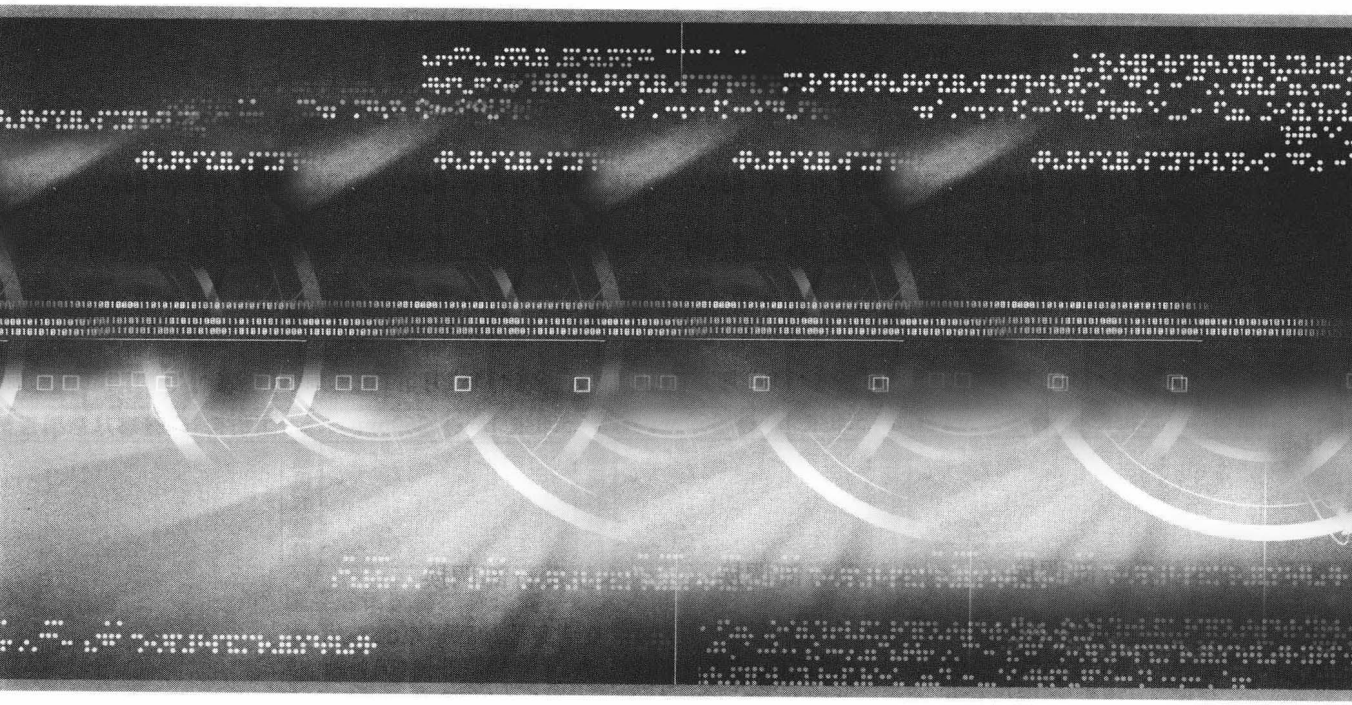
机械工业出版社
China Machine Press

高等院校计算机教材系列

网络编程与分层协议设计

基于Linux平台实现

刘 飚 编著



机械工业出版社
China Machine Press

本书以Linux网络套接字编程和网络分层协议的设计与程序实现为主题，详细介绍如何在Linux平台下进行套接字程序设计，并给出了一个基于分层协议的应用实例，用于模拟Linux网络协议栈中IP层的核心功能——IP报文的分段和重组过程。本书旨在通过具有一定复杂度的应用，切实帮助读者掌握网络协议分层的设计思想和程序实现方法。此外，本书的许多程序示例直接使用了Linux的内核链表和散列链表，以及内核的其他数据结构和多线程等程序设计技巧，通过实际应用的形式有效地衔接了C语言、数据结构、操作系统、计算机网络和网络协议分析等课程的相关内容。

本书可作为高等院校计算机、网络工程、通信工程等专业本科生与研究生“网络程序设计”课程的教材，也可作为相关领域工程技术人员的参考用书。

图书在版编目（CIP）数据

网络编程与分层协议设计：基于Linux平台实现/刘飚编著. —北京：机械工业出版社，2011.7

（高等院校计算机教材系列）

ISBN 978-7-111-35052-1

Ⅰ.网 … Ⅱ.刘 … Ⅲ.① Linux操作系统-高等学校-教材 ② 程序设计-高等学校-教材 Ⅳ.TP316.89

中国版本图书馆CIP数据核字（2011）第114213号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：李 荣

北京诚信伟业印刷有限公司印刷

2011年7月第1版第1次印刷

185mm×260mm · 16.5印张

标准书号：ISBN 978-7-111-35052-1

定价：29.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88379649；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

前　言

随着Internet的迅猛发展，各种各样的网络应用也日益丰富。从传统的Web服务、邮件服务和FTP服务，到实时通信软件、各类网络游戏、流媒体和VoIP等业务，所有这些网络应用都依赖于网络套接字。

套接字是计算机网络最基础的技术之一，通过套接字，Internet上的各类应用就可以使用网络硬件和计算机操作系统所提供的标准机制进行通信。

对于Internet的发展，TCP/IP协议族起着非常重要的作用。其中IP协议成功实现了各类异构网络的互联，成为网络互联协议事实上的行业标准。TCP/IP协议族是典型的分层协议，通常在各类通用操作系统（如UNIX、Linux、Windows）以及路由器等专用操作系统内部实现。对于网络协议分层的概念，各类专著或教材都进行了相关的阐述，然而“纸上得来终觉浅，绝知此事要躬行”，只有通过亲自设计一个分层协议，并编写程序实现基于分层协议的应用，才能真正理解并切实掌握协议分层的思想。

作为优秀的开源操作系统，Linux为学习和掌握网络编程与协议分层的具体实现提供了一个极为方便的学习和应用平台。相比微软的专有Windows操作系统而言，由于Linux的开源特性，其TCP/IP协议栈的全部代码都是公开的，因此通过研究Linux的TCP/IP实现源码，可以快速提高网络软件设计能力。然而，由于TCP/IP协议栈本身比较庞大复杂，并且在Linux系统中以内核代码的形式出现，其中必然会涉及许多操作系统本身的技术内容，这就在一定程度上为掌握TCP/IP协议栈分层的工作原理造成了困难。

为了帮助读者掌握协议分层的设计和实现方法，本书以Linux TCP/IP协议栈中IP协议的核心功能——IP报文分段与重组为重点，参考Linux网络内核2.6.23在IP报文分段和重组中所使用的关键数据结构、处理函数以及处理流程，通过在用户态实现基于应用层消息分段与重组的文件传输功能，成功模拟了IP报文分段与重组的核心处理过程，从而在一定程度上弥补了计算机网络、网络协议分析等课程在讲授网络协议分层方法时纯理论化教学所固有的缺陷。

本书以Linux操作系统为平台，讲解Linux网络编程的基本原理和实现方法，并包含了最新的网络I/O模型EPOLL。另外，本书在内容侧重和程序示例的选取上有别于其他书籍或教材。首先，本书以网络协议为主题，除了介绍IP报文分段和重组模拟程序示例外，还引入了ping程序和FTP客户端程序示例，这3个示例具有一定的复杂度，并且都是紧密围绕TCP/IP协议族中某个具体协议来充分展现网络套接字的应用。其次，本书利用了Linux系统的开源特性，在各类程序示例中，大量直接使用或间接模拟了Linux网络内核所涉及的关键数据结构，有效地体现了数据结构+算法的程序设计思想，便于读者更为具体和深入地了解常用数据结构的应用，从而快速提高网络软件的设计水平。最后，为了便于读者理解书中所给出的Linux下的各个网络程序示例，本书在对网络套接字程序设计和分层协议程序实现进行讲解前，还扼要介绍了掌握本书中各个程序示例所必需的Linux环境编程基础，包括进程、线程、信号、时间与定时以及基本文件操作。因为本书并不是全面介绍Linux编程环境，所以并没有对这些内容进行更深

入和详细的阐述，而是仅围绕后面各章所必需的基本知识进行讲解，这样既突出了主题，又缩减了篇幅，便于读者在不参考其他书籍或教材的情况下就能顺利掌握本书所涉及的内容。

本书共分为9章和3个附录，各章所介绍的内容如下：

第1章介绍TCP/IP网络互联及网络编程所依赖的一些基础概念。

第2章介绍Linux下的C编程环境。本章提供的内容紧扣后续各章所需要的Linux平台下的C编程技术，同时也为不熟悉Linux下的C编程的读者提供了基本的参考。

第3章介绍网络编程所必须掌握的典型知识，这些知识将反复出现在后续章节的程序示例中。

第4章介绍套接字的基础内容，结合第2章所介绍的Linux下的C编程技术，读者可以编写简单的网络应用程序。

第5章介绍套接字的高级特性，主要包括套接字选项、I/O多路复用、非阻塞I/O、信号驱动I/O、事件I/O和原始套接字等内容。

第6章介绍网络协议，包括IP协议、ICMP协议、FTP协议、一个简单的自定义应用协议，以及用于模拟IP报文分段和重组的应用分层协议。本章所介绍的协议内容都将在第7章中通过特定的应用进行实现。

第7章介绍基于ICMP和IP协议，以及原始套接字的ping程序实现方法。

第8章介绍利用流式套接字实现Linux下工作于终端命令行模式的FTP客户端程序的方法。

第9章综合应用本书前面各章所介绍的Linux内核链表、散列链表、线程、用户态下多定时器等内容，重点突出基于分层协议的应用层消息分段和重组的程序实现方法。

附录中给出Linux平台下C/C++语言程序编辑、编译和调试的简洁说明，可供不太熟悉Linux平台下编写C/C++程序的读者编译和调试本书有关程序时进行参考。

为了更好地掌握本书的有关内容，读者应该具有一定的计算机网络基础以及数据结构方面的知识，同时还需要了解Linux下的C语言编程的基本概念。

编者近几年来一直承担研究生“通信协议分析与应用”以及本科生“Linux网络内核源码导读”、“TCP/IP协议”和“Linux网络编程”等课程的教学工作。本书在正式出版前，相关讲义已经在学院内部教学过程中使用了多届次，并根据实际教学中反馈的意见进行了部分修订，收到了良好的效果。

在本书即将出版之际，特别要感谢成都理工大学信息科学与技术学院通信工程系主任李灿平副教授，是他最先鼓励编者尽早整理相关讲义内容并提出了许多中肯而宝贵的意见。另外，感谢通信工程系的王权海老师在本书各个程序示例的调试过程中给予的大力支持。最后，还要特别感谢机械工业出版社华章公司的支持。

热忱欢迎广大读者批评、指导及交流，编者的电子邮箱为：toplb@163.com。

编者

2011年3月

教学和阅读建议

为了更好地掌握本书的内容，先修课程应该包括“C/C++语言程序设计”、“数据结构”和“计算机网络”，并且读者应具有Linux系统最基本的使用经验。此外，由于本书重点在于从实践角度讲解应用层分层协议的设计及实现，因此还可以作为“网络协议分析”等课程的实验教材。

本书学时数安排为40学时左右，理论教学24学时，实验教学16学时。实验教学的内容可以根据实际情况选择本书的相关程序示例，各章的理论教学内容可参考如下安排：

第1章 TCP/IP网络编程基础（1学时）

教学内容：

- 网络和网络互联。
- 客户端/服务器编程模型。

考核要求：通过本章内容，学生应该了解TCP/IP协议的分层结构，理解TCP/IP网络应用相互通信的几个要素，以及客户端/服务器编程的两种模型。

第2章 Linux下的C编程环境（3学时）

教学内容：

- 进程。
- 线程。
- 信号。
- 时间。
- 文件。

考核要求：通过本章内容，学生应该掌握Linux操作系统下父子进程控制、线程同步控制、信号的捕获与处理、间隔定时器的使用和基本文件操作。

第3章 网络编程中常用的典型知识（5学时）

教学内容：

- 结构体大小的计算。
- 数据存储与字节序。
- 常用数据结构。
- 函数指针。
- 校验和。
- 用户态下多定时器的实现。

考核要求：通过本章内容，学生应该掌握利用sizeof计算结构体大小时要注意的问题和对应的解决办法、网络字节序和主机字节序的转换、Linux内核链表和散列链表的使用、校验和原理，了解SNMP多定时器的实现方法。

第4章 基础套接字（4学时）

教学内容：

- 套接字基本概念。

• 流式套接字客户端/服务器编程。

• 并发流式套接字服务器编程。

• 数据报套接字客户端/服务器编程。

考核要求：通过本章内容，学生应该掌握套接字编程的基本知识、并发流式套接字和数据报套接字服务器的编写方法。

第5章 高级套接字（4学时）

教学内容：

• 套接字选项。

• I/O多路复用。

• 非阻塞I/O。

• 信号驱动I/O。

• 事件I/O。

• 原始套接字。

考核要求：通过本章内容，学生应该了解常用套接字选项的用途，掌握select函数实现I/O多路复用的原理和非阻塞套接字I/O的基本概念，另外，还需要了解信号驱动I/O完成数据报套接字编程的方法，掌握事件I/O模型以及原始套接字的基本概念。

第6章 网络协议（3学时）

教学内容：

• IP协议。

• ICMP协议。

• FTP协议。

• 应用层协议示例。

• 分层协议示例——IP报文分段和重组模拟。

考核要求：通过本章内容，学生应该掌握IP协议和ICMP协议的C语言定义、IP报文分段和重组的原理，理解FTP协议的工作原理，了解简单应用层协议的设计，重点掌握分层应用协议示例。

第7章 ICMP协议程序设计（1学时）

教学内容：

• ping程序。

考核要求：通过本章内容，学生应该掌握利用原始套接字处理ICMP消息的程序实现方法。

第8章 FTP协议程序设计（1学时）

教学内容：

• FTP客户端程序。

考核要求：通过本章内容，学生应该掌握利用最基本的流式套接字，实现FTP客户端常用命令程序的方法。

第9章 IP报文分段和重组模拟程序设计（2学时）

教学内容：

• IP报文分段和重组模拟程序。

考核要求：通过本章内容，学生应该正确理解分层协议的处理过程，了解应用层消息分段和重组的程序实现方法。

目　录

第1章 TCP/IP网络编程基础

随着计算机和通信技术的发展，二者的结合也日益紧密，最终计算机网络应运而生。而计算机网络的诞生和发展，又进一步扩大了计算机的应用范围，并促进了计算机技术和通信技术在内的各个领域的飞速发展。由于各类不同的计算机以及由此组成的各类不同的网络千差万别，要将这些系统进行互联，彼此之间的通信就必须遵守一致的约定，即网络协议。Internet通过采用TCP/IP协议族，高效实现了各类不同网络的互联，逐步成为计算机网络协议事实上的工业标准。随着Internet的发展，各类丰富多彩的应用层出不穷，涵盖了人们生活的方方面面，已成为现代人们工作和生活不可缺少的部分。

那么上面所说的各类网络应用程序之间是如何通过Internet进行相互通信的呢？答案就是通过套接字。套接字最初产生于UNIX系统，作为应用和在操作系统内核中运行的TCP/IP协议栈之间的调用接口。套接字分为两类：UNIX BSD套接字和UNIX System V的TLI。因为Sun公司采用了支持TCP/IP的UNIX BSD操作系统，使TCP/IP的应用有了更大的发展，从而促进了套接字在网络软件中的广泛应用，并被引进到以Linux和Windows为代表的系统中，成为开发网络应用软件的强有力工具。本书的目的之一就是通过Linux环境下的C语言网络程序设计过程的详细介绍，帮助读者逐步掌握利用网络套接字和Linux操作系统的基本特性，在通信协议的支持下，实现应用程序间基于Internet的通信。在对网络套接字编程进行深入介绍之前，有必要首先对网络和协议的重要概念和相关内容进行简要说明，从而掌握正确的网络通信模型概念，为顺利学习后续各章的内容打下基础。

1.1 网络和网络互联

计算机网络是由分散在不同位置的计算机通过通信链路连接而构成的网络，这些互联的计算机称为主机和路由器。主机运行各类应用程序，例如即时通信软件、Web浏览器、FTP服务器等，这些应用是网络的实际用户。分散在不同地理位置的计算机网络通过路由器连接而形成Internet，因此路由器的主要职责就是负责在不同的网络之间进行数据转发，从而保证不同主机之间能够跨越多个互联网络进行数据通信。图1-1描述了通过路由器连接两个网络所构成的网间网Internet。

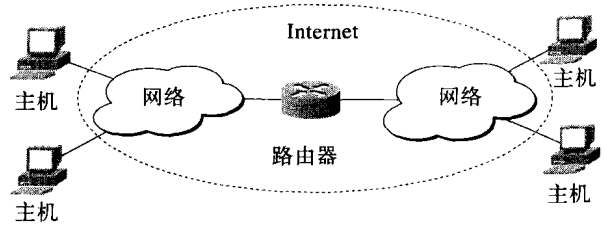


图1-1 Internet

1.1.1 ISO/OSI网络模型和TCP/IP协议栈

为了让Internet上不同厂家、不同型号和不同年代的计算机之间能够彼此通信，这些系统就要彼此开放，且遵循同样的标准，这个标准就是TCP/IP协议体系。在TCP/IP协议族中，网络的体系结构按照不同的功能分为5层，分别是物理层、数据链路层、网络层、传输层和应用层。另外，根据国际标准化组织ISO所定义的OSI（Open System Interconnection），网络结构按照不同的功能又分为7层，分别为物理层、数据链路层、网络层、传输层、会话层、表示层与应用层。图1-2描述了这两者之间的大体关系。

在TCP/IP的5层网络模型中，IP层属于第3层（网络层），TCP层属于第4层（传输层），能在主

机间提供可靠的数据传输。传输层除了TCP协议外，还有另外一个协议叫UDP，UDP提供不可靠的数据传输。

对于网络的分层模型而言，其中最关键的内容就是各层所定义的协议。协议是指各层通信实体就本层所交换数据的表示方法和含义达成的一致约定。图1-2中物理层、数据链路层、网络层和传输层等都有自身的协议约定，通信双方只要遵守这个约定，就可以相互理解对方发送数据的含义。另外，上述四个协议层通常都在操作系统如Linux的内核中加以实现，而应用层通常由用户态下的程序来完成，并且对于应用层而言，往往也有自身的应用协议。例如Internet上的标准服务WWW使用的是HTTP协议，FTP服务使用的是FTP协议，电子邮件使用的是POP3和SMTP协议。此外各类IM（即时通信）应用也都有自身的私有协议。图1-3给出了TCP/IP协议族的核心协议。

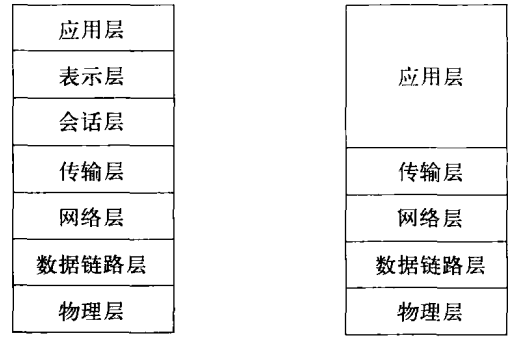


图1-2 OSI的7层模型与TCP/IP的5层模型之比较

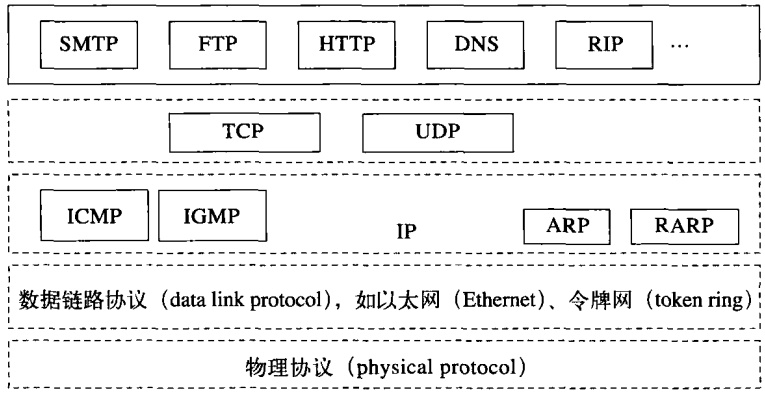


图1-3 TCP/IP协议族

通常，图1-3中应用层之下的各层协议都在操作系统的内核中实现，而应用层协议由用户态程序实现。

1.1.2 IP地址和端口

在TCP/IP网络中，如果有两个网络应用程序要相互通信，首先需要知道的信息就是IP地址（IP address）。IP地址在IP协议中定义为一个无符号的32位比特数，如0xC0A8008D。为了便于人们阅读，通常使用点分十进制数字形式来表示一个IP地址，于是上述无符号32位比特数0xC0A8008D又可以表示为192.168.0.141。

IP地址仅用于标识参与通信的源和目的主机，即所谓的主机-主机通信。作为网络层协议，IP只负责将数据从源主机发送到目的主机。然而通信实际发生在位于网络中不同主机中的应用或进程，因此还需要有另外一个重要信息来标识主机中的通信进程，这个信息就是端口号（port number）。端口号是一个16位比特的无符号数，范围为0~65 535，其中0~1023属于众所周知端口，由IANA（Internet Assigned Numbers Authority）负责分配和控制；1024~49 151属于可注册端口，若需使用，应向IANA注册并由IANA公开，以免发生重复；49 152~65 535为动态端口，IANA未对该范围内的端口进行规定，可供任何进程使用。端口号代表主机上运行的进程，不同进程拥有不同的端口。

不论是源端还是目的端，都会有IP地址和端口，因此由源端建立一条连接到目的端，需要的信息包括：源IP、源端口、目的IP和目的端口共4个参数。这4个参数又称为套接字对（socket pair），它标识了通过TCP/IP协议进行通信的双方。图1-4描述了参与通信的套接字对，其中（202.115.128.33, port1, 172.168.3.21, port3）构成一个通信套接字对，而（202.115.128.33, port2, 172.168.3.21, port4）也构成一个通信套接字对。

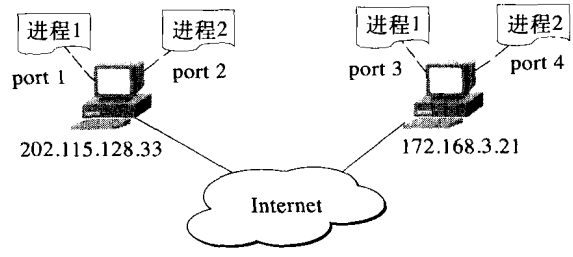


图1-4 套接字对

1.2 客户端/服务器编程模型

客户端是指运行在本地主机上的一个程序，该程序请求服务器程序提供某种服务，例如文件传输，或者Web页面等。客户端程序使用远端主机的IP地址和对应的众所周知的端口与远端主机上的服务器程序进行通信，这一过程称为主动打开（active open）。客户端程序根据用户需要启动，获得所请求的服务后就结束运行，此时客户端主动结束通信（active close）。

服务器是指运行在远端主机上的一个程序，该程序向客户端提供某种服务。当服务器程序启动后，将一直等待客户端发来请求，服务器自身并不会主动发起请求，这一过程称为被动打开（passive open）。另外，通常服务器程序一旦启动后，将一直运行下去，除非有异常情况发生而导致其中止。

1.2.1 循环式服务器

所谓循环式服务器，指服务器一次只处理一个请求，当处理结束向客户端发回应答之后，才能接受下一个客户端请求。通常使用UDP协议的服务器工作于循环模式，如图1-5所示。

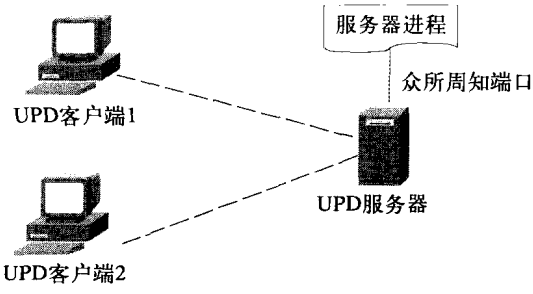


图1-5 循环式服务器

1.2.2 并发式服务器

所谓并发式服务器，指服务器可以同时处理多个客户端发来的请求，常用于TCP协议服务器。对于TCP协议服务器，客户端和服务器之间将建立一个连接用于数据交换，该连接在数据传输过程中一直保持，直到通信结束后关闭。

并发式服务器不能仅靠使用一个众所周知端口来完成连接过程，因为服务器需要为每个连接都分配一个端口，而并发式服务器又要同时服务于多个客户端。因此除了一个众所周知的端口用于监听客户端的连接请求外，还必须为每个已经建立连接的客户端分配一个临时端口用于进行数据交换。通常TCP服务器在众所周知端口上工作于被动打开模式，当收到客户端发来的连接请求后，服务器

为此连接分配一个临时端口，于是用于监听的端口将被释放出来继续监听下一个请求。Linux下通常采用父子进程实现此工作方式，负责接收客户端连接请求的称为父进程（parent server），它工作于众所周知端口，子进程（child server）工作在临时端口。如图1-6所示，client1和client2分别都和TCP服务器建立了各自的TCP连接，TCP服务器创建了2个子进程child server，分别通过两个临时端口port1和port2与两个客户端进行数据传输。

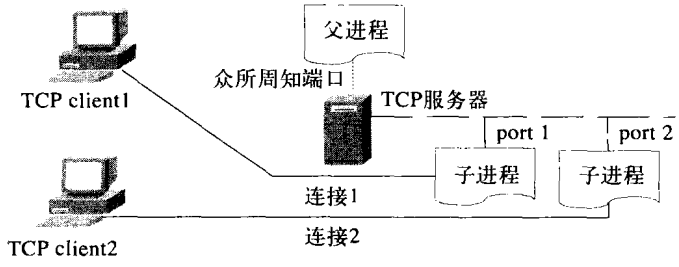


图1-6 并发式服务器

1.3 本章小结

本章简要介绍了TCP/IP协议栈和ISO/OSI网络参考模型的基本内容，初步说明了协议分层的概念，在后续的第6章中，还将实际动手定义一个应用层的分层协议，并在Linux环境下应用本章所介绍的循环式服务器模型完成一个基于该分层协议的应用。

对于TCP/IP网络通信程序，一个最重要的概念就是套接字对，即源IP、源端口、目的IP和目的端口，主机之间的通信就是通过建立在一对套接字对之间的逻辑通道进行的。

除了协议分层概念外，本章还介绍了客户端/服务器编程模型，对于TCP协议，在服务器端通常可以使用并发式服务器进行编程，而对于UDP协议，通常在服务器端使用循环式服务器编程。

习题

1. OSI网络模型和TCP/IP协议栈之间有着怎样的关系？
2. TCP/IP网络中，进行相互通信的应用程序必须知道哪些信息？
3. 为什么要使用端口号，其作用是什么？
4. 客户端/服务器编程中的主动打开和被动打开的含义是什么？
5. 循环式服务器和并发式服务器在工作原理上有什么不同？
6. 并发式服务器中所使用的监听端口和数据传输端口是同一个端口吗？请举例说明你所知道的众所周知端口。

第2章 Linux下的C编程环境

本章简要介绍后续各章节中将要使用的Linux下的C编程的相关重要内容，包括进程、线程、信号、时间和计时以及基本文件操作。为了突出本书的重点，本章并不准备对Linux下的C编程环境进行全面介绍，仅围绕本书后续各章节中所需要的Linux下的C编程知识进行扼要说明，以帮助读者在不参考其他书籍的情况下也能顺利阅读后续章节中的各类示例程序。

2.1 进程

一个正在运行的程序称为进程。例如在屏幕上正打开两个终端窗口，则说明同一个终端程序正在作为两个进程而同时执行，而每个终端窗口又都在执行shell，则每个shell又是另外一个进程。当通过shell命令启动一个程序后，对应的程序将作为新的进程启动。网络通信程序的设计通常都需要在一个应用中使用多个互相协作的进程（或线程）来同时完成多个任务，以提高应用的效率，从而以尽量短的时间响应网络中大量用户的同时请求。

本节将对应用如何创建进程，父子进程间怎样保证同步，以及子进程终止后如何善后等相关函数进行简明扼要的说明，更详细的内容请参考其他有关书籍，另外也可以通过随机的man手册获得各个函数的详细内容。

2.1.1 进程标识

每一个Linux下的进程都要分配一个唯一的进程标识（pid），pid是一个16比特的整数。此外，除了init进程外，每一个进程都有一个父进程。

当在C程序中使用进程标识时，通常用pid_t类型来声明进程标识变量，并通过getpid()系统调用获得一个进程自身的pid，而通过getppid()系统调用获得其父进程标识ppid。

例如，可以通过下面的程序2.1获得一个进程及其父进程的标识。

程序2.1 getpid.c

```
1 #include <stdio.h>
2 #include <unistd.h>
3 int main ()
4 {
5     printf ("The process ID is %d\n", (int) getpid ());
6     printf ("The parent process ID is %d\n", (int) getppid ());
7     return 0;
8 }
```

参考附录B.1编译并运行程序2.1，结果如下：

```
$ gcc getpid.c -o getpid -std=gnu99
$ ./getpid
The process ID is 31997
The parent process ID is 25958
```

Linux下可以通过ps命令查看当前有哪些进程正在运行，其中ps命令具有很多选项，各个选项的含义可以参考随机man手册。例如下面的ps命令将用列表表示属于当前Linux用户进程的相关信息：

```
$ ps -a -o pid,ppid,command
```

```
  PID   PPID    COMMAND
28366   28333  ./tcpserver 9000
28393   28366  ./tcpserver 9000
28914   27018  ps -a -o pid,ppid,command
```

2.1.2 创建进程

Linux下有多种创建进程的方法，本节仅对其中的fork系统调用方式进行扼要介绍。

当程序执行fork时，一个称为子进程的进程就被当前进程创建了，并且该子进程就是当前父进程的复制品。父进程则继续从派生子进程处执行，而子进程也从其产生处开始执行。

因为每个进程都有唯一的标识pid，而子进程是一个新产生的进程，一定具有不同于父进程的pid，所以就可以通过pid来区分父进程和子进程。一个区分的方法就是通过getpid获知当前进程的pid值。另外，也可以根据fork系统调用的返回值来判断，即父进程执行完fork后的返回值就是子进程的pid，而在子进程中该返回值为0，显然没有任何进程的pid会等于0，于是就可以在程序中轻易区分当前到底是父进程还是子进程正在执行。

程序2.2说明了如何利用fork创建子进程。

程序2.2 forkprocs.c

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 int main ()
5 {
6    pid_t child_pid;
7    int i = 1;
8    printf("the main program process ID is %d\n", (int)getpid());
9    child_pid = fork(); /* 创建子进程 */
10   if (child_pid != 0) {
11     i = 0;
12     printf("this is the parent process, with id %d and i = %d\n",
13           (int)getpid(), i);
14     printf("the child's process ID is %d\n", (int)child_pid);
15   } else
16     printf("this is the child process, with id %d and i = %d\n",
17           (int)getpid(), i);
18   return 0;
19 }
```

参考附录B.1编译并运行程序2.2，结果如下：

```
$ gcc forkprocs.c -o forkprocs -std=gnu99
$ ./forkprocs
the main program process ID is 32148
this is the child process, with id 32149 and i = 1
this is the parent process, with id 32148 and i = 0
the child's process ID is 32149
```

可以看出，父进程和子进程各自占有完全独立的地址空间，因此尽管子进程中将变量i的值改变为0，而父进程的同名变量的值仍然为1。

2.1.3 终止进程

通常进程可以通过两种方式终止，即调用exit函数退出，或者程序的main函数执行return语句退出。每个进程终止时都有一个退出码返回给其父进程，退出码由终止进程作为参数传递给exit函数，或者作为main函数的return语句所返回的值。

当进程收到信号后，也可能会意外终止。例如Linux下可以通过kill命令向一个进程ID为28393的进程发送SIGKILL信号，从而终止正在运行的进程28393：

```
$ kill -s SIGKILL 28393
```

由于进程收到了一个未设置对应处理程序的信号，默认的处理通常就是终止进程。因此当一个网络应用进程由于I/O端口阻塞时，若收到了意外的信号，则通常会终止应用的运行。

2.1.4 僵死进程

当用fork系统调用创建子进程后，子进程就独立于父进程开始自身的运行。当子进程先于父进程执行完毕后，尽管此时子进程已经不会再活动了，但是代表子进程的数据结构并不会立刻从进程表中销毁。因为子进程的退出代码需要保存起来以供父进程随后调用wait时使用，此时子进程变成了不再活动的进程，也就是通常所说的“僵死”进程。僵死进程虽然不再活动，但是仍然占用一定的系统资源，直到父进程调用wait获得子进程的终止信息后，才被最终予以释放。

程序2.3演示了僵死进程是如何产生的。

程序2.3 zombie.c

```
1 #include <stdlib.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 int main ()
5 {
6   pid_t child_pid;
7   /* 创建子进程 */
8   child_pid = fork ();
9   if (child_pid > 0) {
10    /* 父进程休眠60秒 */
11    sleep (60);
12  } else {
13    /* 子进程立即退出 */
14    exit (0);
15  }
16  return 0;
17 }
```

在Linux的终端中编译并运行程序2.3，然后打开另外一个终端，执行如下命令：

```
$ ps -a -o pid,ppid,s=state,command
   PID    PPID   state   COMMAND
   3116   28333  S       ./zombie
   3117   3116   Z       [zombie] <defunct>
   3132   27018  R       ps -a -o pid,ppid,state=state,command
```

可以看到pid为3117的子进程派生自pid为3116的父进程，并且此时该子进程进入了僵死状态，state列显示为Z。这是因为父进程先休眠60秒，在休眠期间子进程得以执行并立即执行exit，先于父进程退出，此时父进程尚未对子进程进行善后处理，因此已经退出的子进程仍然占有少量的系统资源。

2.1.5 子进程退出的异步善后处理

为了避免僵死进程长期存在，父进程希望知道子进程什么时候结束运行。可以通过让父进程调用wait或waitpid暂时停止执行，等待子进程终止运行后，再继续执行父进程，函数原型如下：

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *stat_loc);
```